Computer Microoperations

- A microoperation is an elementary operation performed with the data stored in registers.
- The microoperations most often encountered in digital computers are classified into four categories:

1- Register transfer microoperations: transfer binary information from one register to another.

2- Arithmetic microoperations: perform arithmetic operations on numeric data stored in registers.

3- Logic microoperations: perform bit manipulation operations on non- numeric data stored in registers.

4-Shift microoperations: perform shift operations on data stored in registers.

Basic arithmetic microoperations

- Addition
- Increment
- * Subtraction
- * Decrement

- Add microoperation

  $R3 \leftarrow R1 + R2$

- To implement this statement with hardware we need three registers and the digital component that performs the addition operation
- Subtraction is most often implemented through complementation and addition.
- Instead of using the minus operator, we can specify the subtraction by the following statement:

  $R3 \leftarrow R1 + R2' + 1$

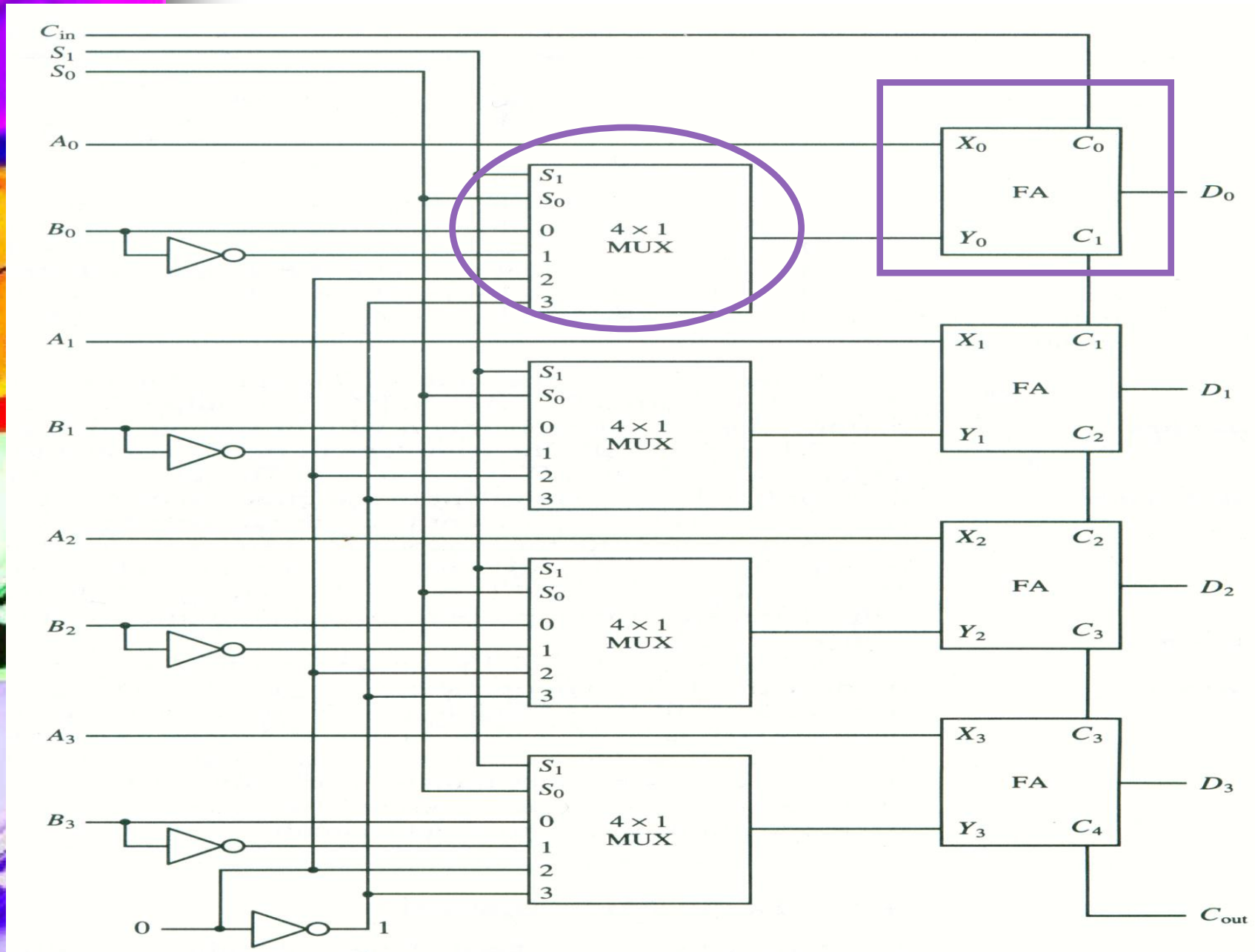
Arithmetic Microoperations

Symbolic designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2's complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

Binary Adder

- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition.
- A full-adder is the digital circuit that forms the arithmetic sum of two bits and a previous carry.
- A binary adder is the digital circuit that generates the arithmetic sum of two binary numbers of any lengths.
- Binary Adder-Subtractor

Arithmetic Circuit Implementation



Functions performed by the circuit

Select			Input	Output	microoperation
S1	S0	Cin	Y	D=A+Y+ Cin	
0	0	0	B	D=A+B	Add
0	0	1	B	D=A+B+1	Add with carry
0	1	0	B'	$D=A+B'$	Subtract with borrow
0	1	1	B'	$D=A+B'+1$	Subtract
1	0	0	0	D=A	Transfer A
1	0	1	0	D=A+1	Increment A
1	1	0	1	D=A-1	Decrement A
1	1	1	1	D=A	Transfer A

- When all 1's are inserted into the Y inputs
- A number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111).
- Adding a number A to the 2's complement of 1 produces

F = A + 2's comp. of 1 = A - 1. (decrement)

- When C_{in}= 1, then D = A - 1 + 1= A,

Logic Microoperations

Logic Microoperations

Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1's complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

Logic Microoperations

- The + symbol has two meanings; it will be possible to distinguish between them by noting where the symbol occurs.
- When the symbol + occurs in a microoperation, it will denote an arithmetic plus.
- When it occurs in a control (or Boolean) function, it will denote an OR operation

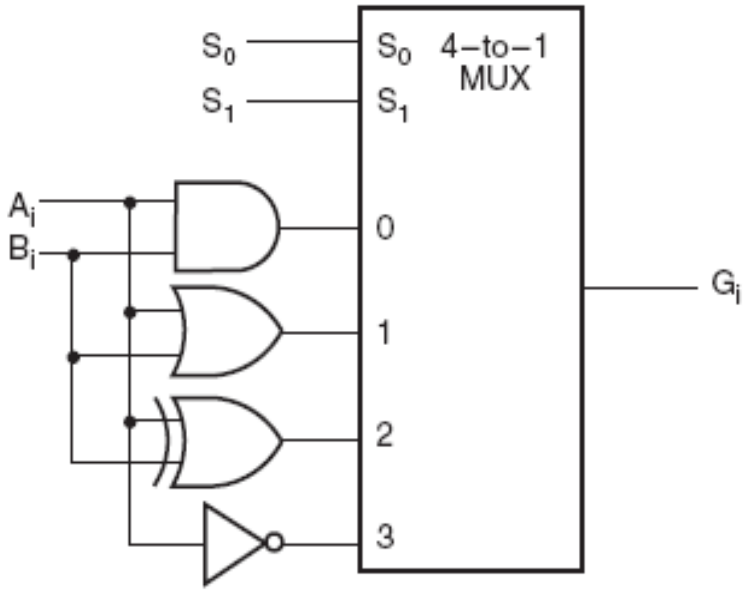
For example,

- P + Q: R1← R2 + R3, R4 ← R5 V R6

- The + between P and Q is an OR operation between two binary variables of a control function.
- The + between R2 and R3 specifies an add microoperation.
- The OR microoperation is designated by the symbol V between registers R5 and R6.

Hardware Implementation of logic microperations

- It requires that logic gates be inserted for each bit or pair of bits in the registers to perform the logic function.
- Although there are 16 logic microoperations, most computers use only four—AND, OR, XOR (exclusive-OR), and complement— from which all others can be derived.



(a) Logic Diagram

S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

(b) Function Table

Shift Microoperations

- *Used* for serial transfer of data.
- They are also used in conjunction with arithmetic, logic, and other data-processing operations.
- The contents of a register can be shifted to the left or the right.
- There are three types of shifts:
  a) Logical
  b) circular
  c) arithmetic

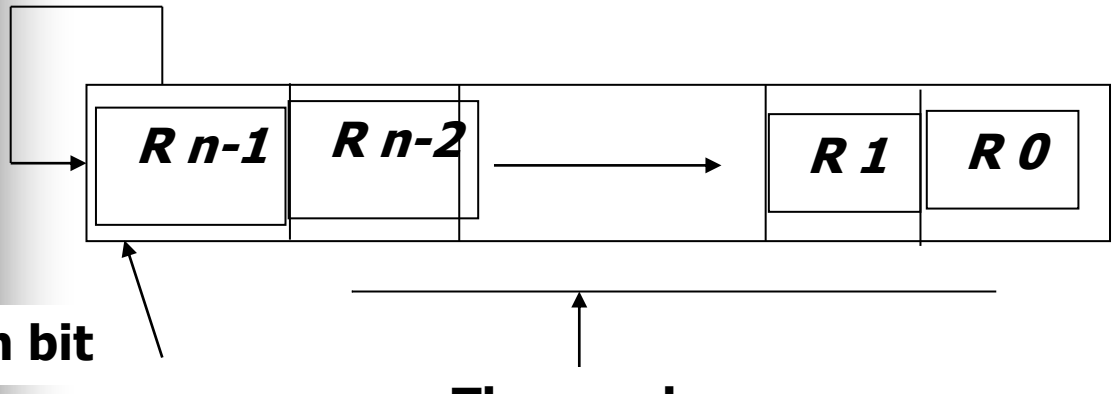
Symbolic designation	Description
R ← shl R	Shift-left register *R*
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left *R*
R ← cir R	Circular shift-right *R*
R ← ashl R	Arithmetic shift-right *R*
R ← ashrR	Arithmetic shift-right R

An *arithmetic* shift:

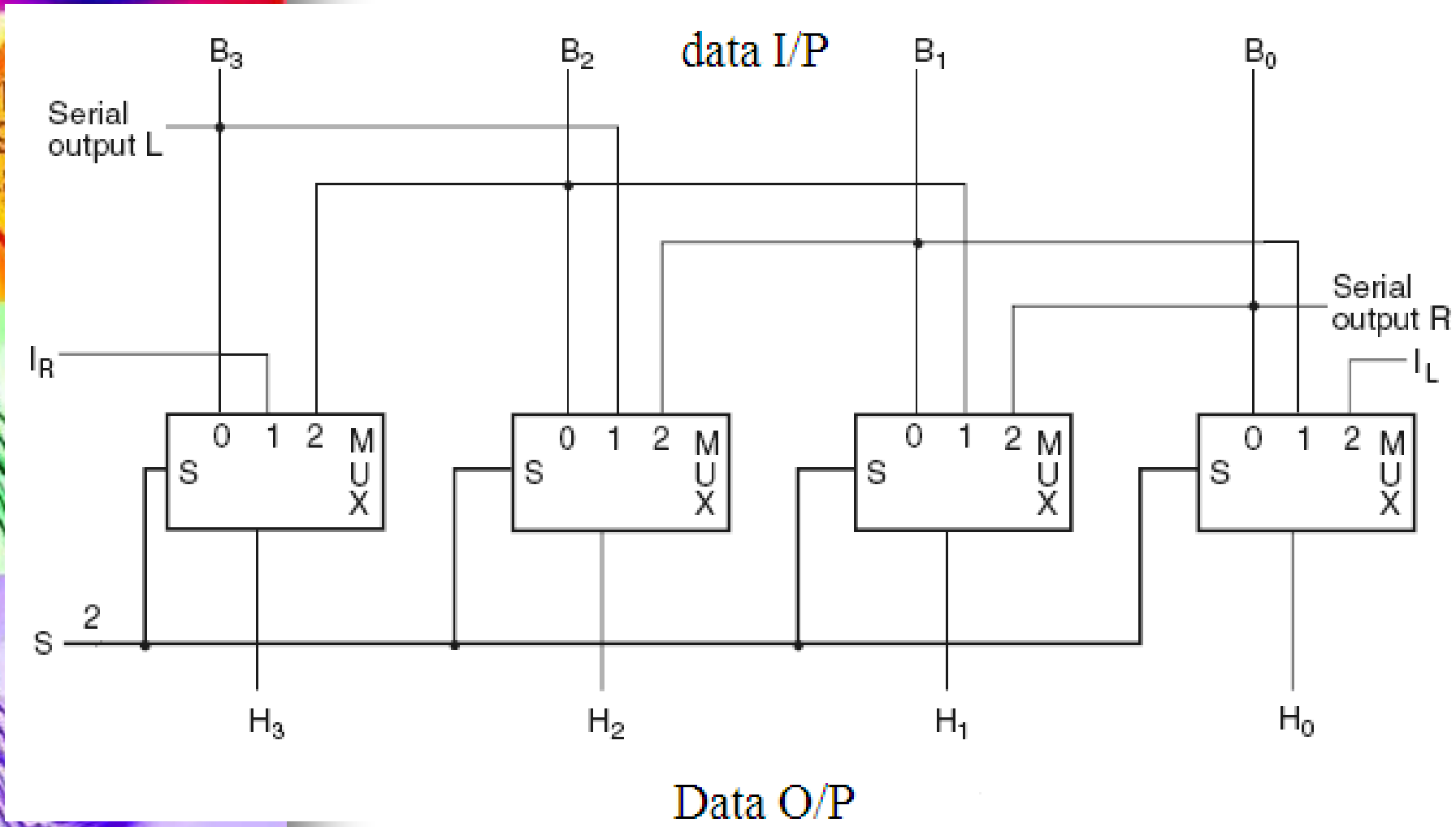
- Is a microoperation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.

Notes:

- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

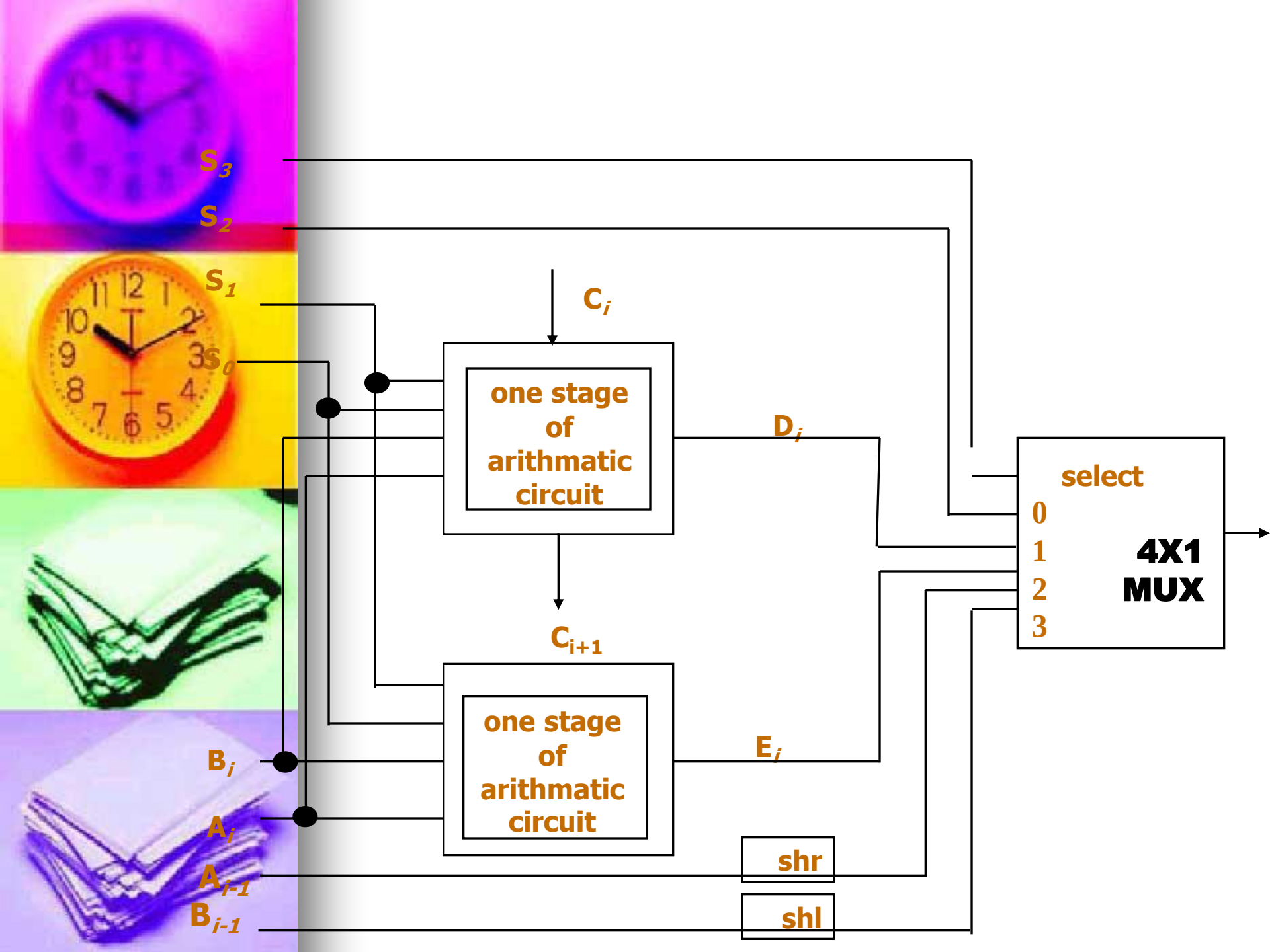


Hardware Implementation of logic microperations



Arithmetic-Logic-Shift Unit

- Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.
- To perform a microoperation, the ALU performs an operation and the result of the operation is then transferred to a destination register.
- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables.


S_3
S_2
S_1
S_0
C_i
one stage of arithmatic circuit
D_i
select
0
1
2
3
4X1 MUX
C_{i+1}
one stage of arithmatic circuit
E_i
B_i
A_i
A_{i-1}
B_{i-1}
shr
shl

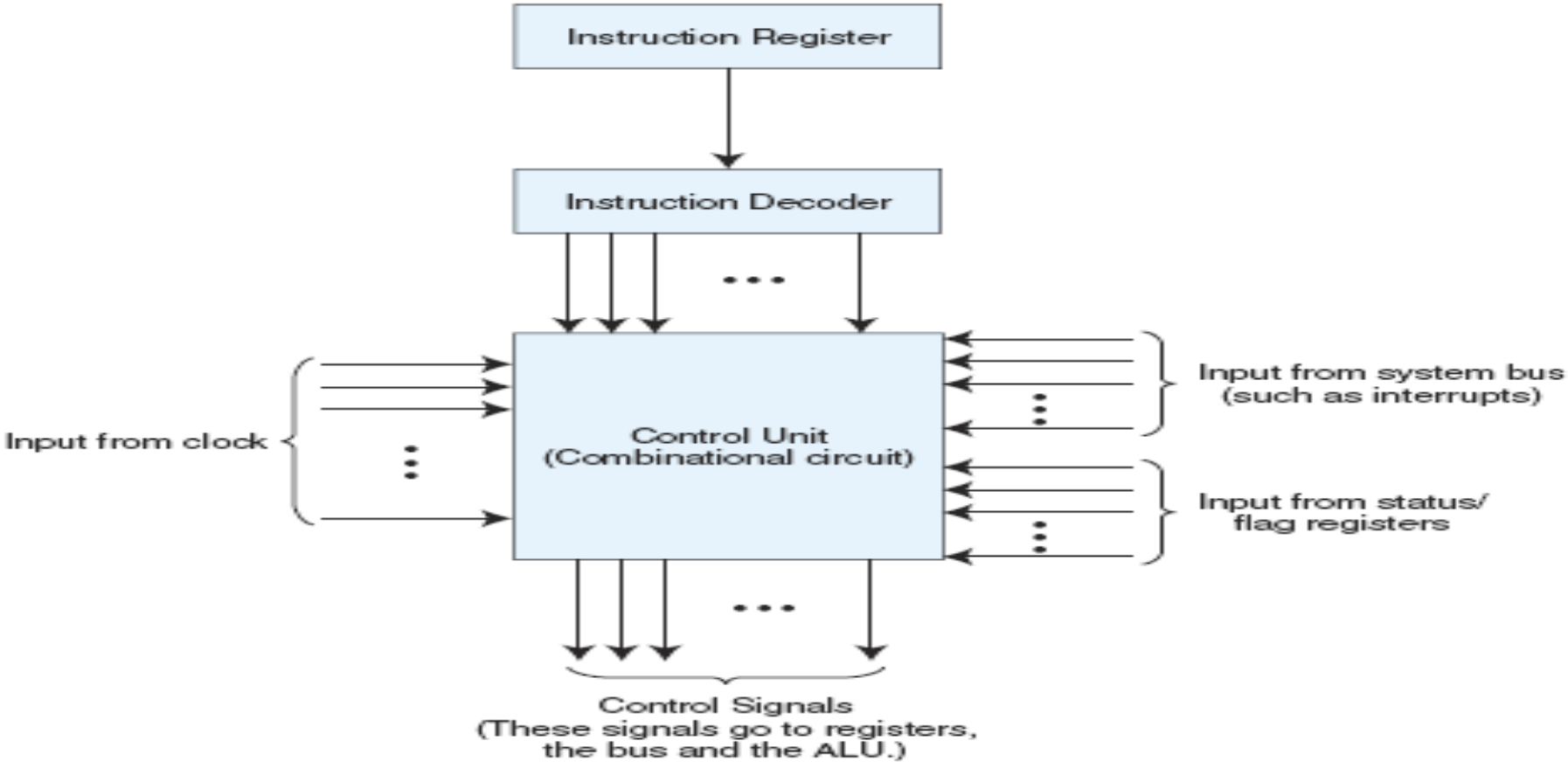
Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + \underline{B} + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1's complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	0	0	X	$G = A \wedge B$	AND
1	0	1	X	$G = A \vee B$	OR
1	1	0	X	$G = \underline{A} \oplus B$	XOR
1	1	1	X	$G = A$	NOT (1's complement)

A DISCUSSION ON DECODING: HARDWIRED VS. MICROPROGRAMMED CONTROL

- In reality, there must be control signals to assert lines on various digital components to make things happen as described.
- For example, when we perform an *Add* instruction in assembly language, we assume the addition takes place because the control signals for the ALU are set to "add" and the result is put into the AC.
- The ALU has various control lines that determine which operation to perform.
- The question we need to answer is, "How do these control lines actually become asserted?"
- You can take one of two approaches to ensure control lines are set properly.

The first approach (hardwired control)

- Is to physically connect all of the control lines to the actual machine instructions.
- The instructions are divided up into fields, and different bits in the instruction are combined through various digital logic components to drive the control lines.



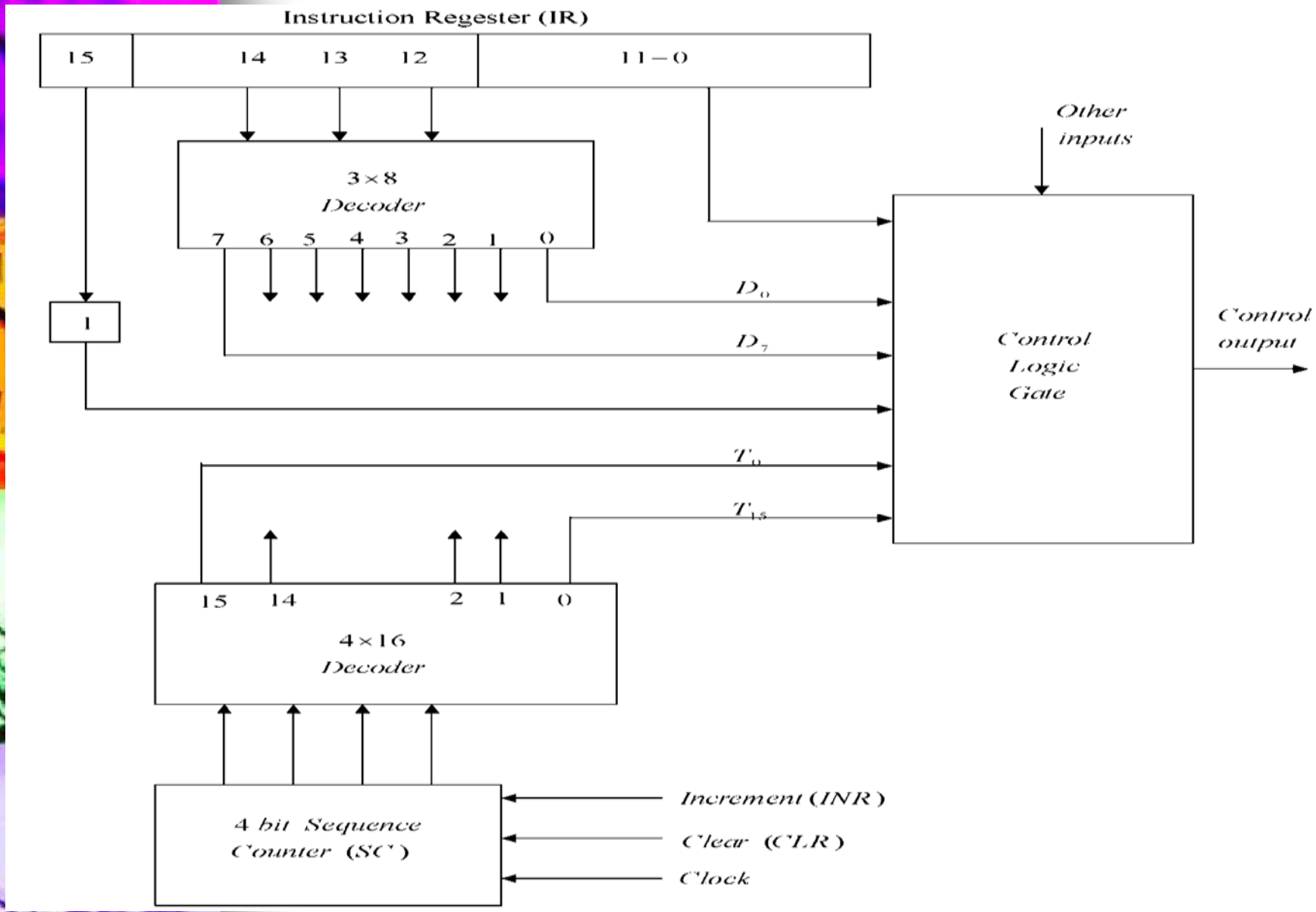
For example:

- A 4-to-16 decoder could be used to decode the opcode.
- By using the contents of the IR register and the status of the ALU, this unit controls the registers, the ALU operations, all shifters, and bus access.

- ●*The advantage* of hardwired control is that it is very fast.

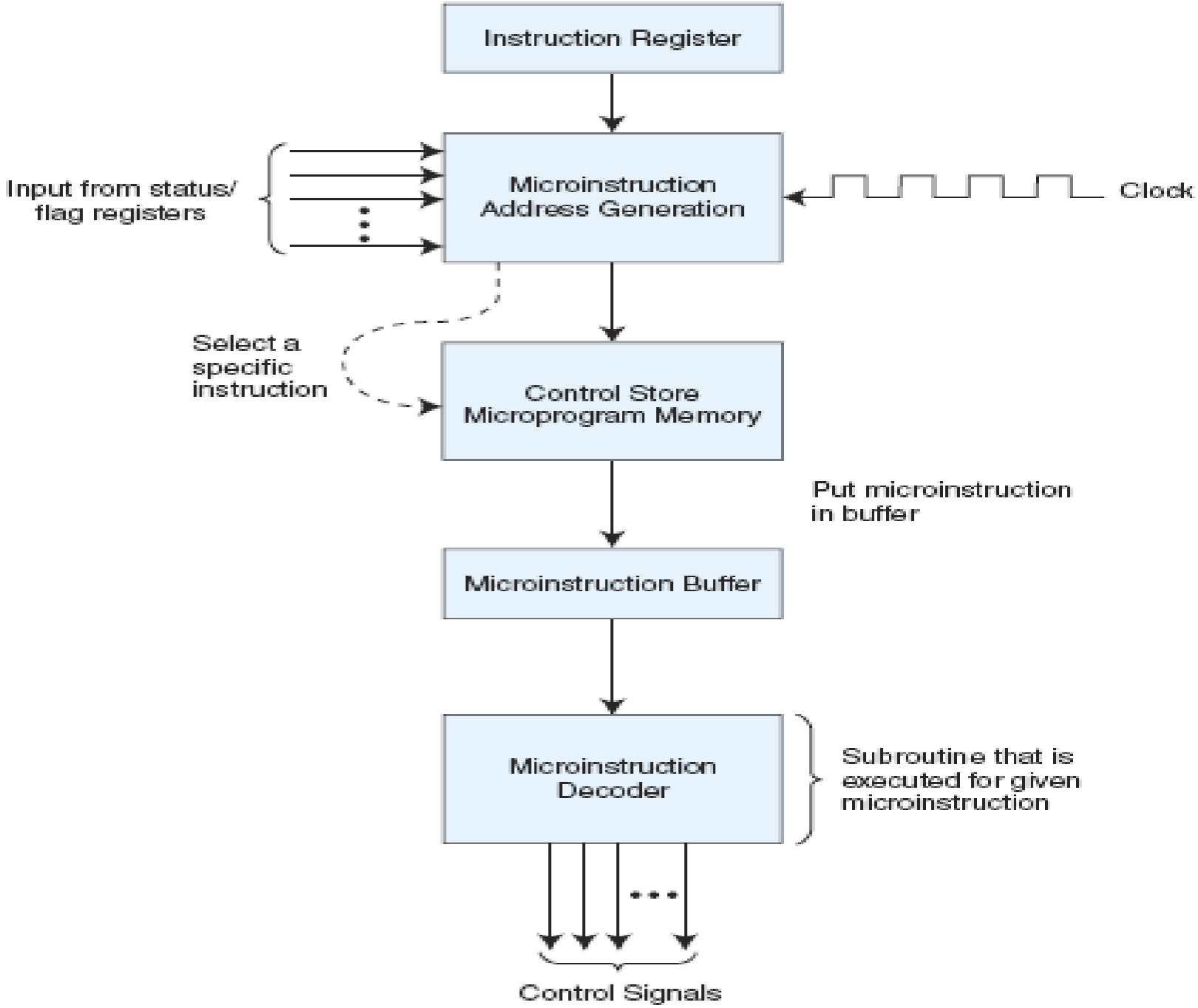
- ●*The disadvantage* is that the instruction set and the control logic are directly tied together by special circuits that are complex and difficult to design or modify.

- If someone designs a hardwired computer and later decides to extend the instruction set, the physical components in the computer must be changed. This is prohibitively expensive, because not only must new chips be fabricated but also the old ones must e located and replaced.


Instruction Regester (IR)
15
14
13
12
11 – 0
3 × 8
Decoder
7
6
5
4
3
2
1
0
1
D_0
D_7
Other
inputs
Control
Logic
Gate
Control
output
T_0
T_{15}
15
14
2
1
0
4 × 16
Decoder
4 bit Sequence
Counter (SC)
Increment (INR)
Clear (CLR)
Clock

The Second approach (*microprogramming)*

- uses software for control, and is illustrated in Figure
- All machine instructions are input into a special program, the *microprogram*, to convert the instruction into the appropriate control signals.
- The microprogram is essentially an interpreter, written in *microcode*, that is stored in *firmware* (ROM, PROM, or EPROM), which is often referred to as the *control store*.
- This program converts machine instructions of zeros and ones into control signals.
- Essentially there is one subroutine in this program for each machine instruction.


Instruction Register
Input from status/
flag registers
Microinstruction
Address Generation
Clock
Select a
specific
instruction
Control Store
Microprogram Memory
Put microinstruction
in buffer
Microinstruction Buffer
Microinstruction
Decoder
Subroutine that is
executed for given
microinstruction
Control Signals

- The advantage of this approach is that:
- If the instruction set requires modification, the microprogram is simply updated to match—no change is required in the actual hardware.
- Microprogramming is flexible, simple in design, and lends itself to very powerful instruction sets.
- Microprogramming allows for convenient hardware/software tradeoffs: If what you want is not implemented in hardware (for example, your machine has no multiplication statement), it can be implemented in the microcode

- →<u>The disadvantage to this approach is that:</u>
-
- All instructions must go through an additional level of interpretation, slowing down the program execution.
- In addition to this cost in time, there is a cost associated with the actual development, because appropriate tools are required.